Waste sorting error detection

Thomas Diaconu

Paprec France

Academic Supervisor: Maria Zuluaga, professor at Eurecom
Company Supervisor: Pierre Allain, director of collection department

Abstract

Paprec is a major French company in recycling industry. Every day, trucks collect waste all over France. In order to optimize recycling, it is important to identify the type of waste collected. Indeed, respecting waste sorting is very significant in recycling. Each truck is equipped with cameras, which screen capture of the different waste it contains. Screens are taken at every container discharging. In one day, a truck collects waste around 50 times and at each collect, the camera disposal takes approximately six pictures. Then, these pictures are registered in a flash drive in order to be checked by a team. This team has to check waste type and to signal the noncompliance (e.g. glass bottle mixed with plastic or synthetic materials). These mistakes are reported to the client in the form of a de-rating sheet. This project consists in using artificial intelligence in order to automate this error detection. Several models will be implemented and analyzed by their performances.

Contents

1 Paprec

1.1 Company overview

Paprec is a French company specialized in waste collection and recycling founded in 1994. It is now a leading company in waste industry. The company is the leader in waste recycling and 3rd in waste treatment. Since its creation, the group has constantly grown. Indeed, there were 45 employees in 1994. Now the company has 10.000 employees.

Moreover, the strength of Paprec is that they have numerous private clients (about 45.000). The company has also many sites in France and Switzerland (about 200). Thanks to all of these partners and to this presence all over the country, the company treats about 12 million of tonnes of waste per year. Paprec treats the selective collection of one out five French people.

1.2 Informatic department

There are many departments at Paprec, especially the **IT department**. It is in charge of informatic and computer science projects. This department has three main missions:

- maintain the **infrastructure**: servers, VPN, network devices management, IT purchases, management ...
- ensure the operation of intern applications
- develop new applications destined to the recycling industry and for the only use of Paprec

This department works like a software and consulting company that has the goal to ensure the proper functioning of the IT in Paprec.
This department may need to collaborate with other departments. In the context of this project, the IT department collaborate with the **collection department**. We will explain this collaboration in the next section.

2 Project description

2.1 Demand

Let's firstly introduce the **collection department**. The collection department is the division in charge of waste collection. It manages the waste collection with trucks and also negotiations with municipalities in order to allow Paprec trucks to collect waste.
Observing that waste sorting isn't always respected, the department has to regularly verify the respect of waste sorting rules. To do that, each truck is equipped with cameras, which screen capture of the different waste it contains. Screens are taken at every container discharging. In one day, a truck collects waste around 50 times and at each collect, the camera disposal takes approximately six pictures. Then, these pictures are registered in a flash drive in order to be checked by a team. Then, the team checks waste type manually and penalize organizations that don't respect the waste sorting (e.g. glass bottle mixed with plastic or synthetic materials) to signal the noncompliance. The departments demand

is to automatically compute this detection: manually checking takes too much time. This project consists in solving this problem: **automate this waste detection as well as possible**.

2.2 Expectations

The expected product is a program which consists in detecting undesirable waste. The first approach is to detect if one of the three following type of waste is present in the collection: **glass**, **bags**, **wood**. The detection must be as precise as possible. In other words, if the program detects a type of waste, we must be almost sure that the detection is correct. Indeed, this program aims to help the department to signal the noncompliance in waste sorting to the client. It is obvious that it is important to signal a correct abuse rather than a false one.

2.3 Approach

In this project we are going to use artificial intelligence and Machine Learning tools. The main objective is to detect three types of waste and to classify them. Thus, we are facing a classical problem of **image classification** which is based on Machine Learning and Deep Learning tools. There are several approaches that can be adopted to solve these kind of problems: logistic regression decision tree, random forest, neural networks, etc. The most common is **neural networks** which we are going to use here. This choice has been made for different reasons. Firstly, the provided solution was a neural network, which directly leads my choice to this tool. Secondly, because neural networks commonly have higher accuracy in this kind of problems (better results, no need of feature extraction). Therefore, our approach will be as following: firstly, we will use a pre-trained neural network, evaluate its performances and optimize some parameters. Then we will implement our own model.

3 First approach: pre-trained model

3.1 CortexIA model

3.1.1 About the company

CortexIA is a Swiss start-up based in Châtel-Saint-Denis, near Lausanne in Switzerland. This company actively works on the model of the **smart city**, especially in terms of **cleanliness**. By using computer vision and Deep Learning, CortexIA product is capable of detect and classify waste on public roads. This model has been adopted in important Swiss cities as Zürich, Geneva or Fribourg. It has been even proposed by Cedric Villani to adopt this approach in Paris [1].

In order to get some results, Paprec (the collection and IT departments in particular) asked a collaboration with CortexIA in order to detect waste sorting error. CortexIA trained its model with Paprec photos and provided the output neural network. The first step of the project is to understand their model, call it and execute it on a set of photos in order to compute its performances.

3.1.2 Description of the model

The model is based on Deep Learning and **CNNs** (Convolutional Neural Networks), especially R-CNNS [5]. Their approach is very similar to **OverFeat model**. The classification architecture has been replaced by GoogLeNet. For localization, the classifiers layers have been replaced by a regression network and the image is analyzed at each spatial location and scale. Then, the regression network makes predictions at each spatial location of the picture. Object bounding boxes are obtained by running the classifier and the regression network for all locations and scales possibles of the image. After recomputing the final regression layers, the final output layer obtained provides 4 units. These 4 units are essential because they correspond to the coordinates of the bounding box of the detected object.

For the implementation, they used an open source implementation of OverFeat on Tensorflow as a starting point. Some modifications were done in order to compute the multi classification and also a change concerning the resolution. The model is adjusted on **Tensorflow** for 350.000 iterations with a batch size of 16. A validation is computed every 2000 iterations.

Now that we described the CortexIA model, we will present our way to load the neural network, and to convert the predictions output into usable data.

3.2 Algorithm presentation

3.2.1 Steps and code structure

As we said before, CortexIA provided the pre-trained neural network. This neural network has been given in the form of ckpt files (checkpoint files). These files contain the exact value of the model parameters. However, there isn't any information about the calculus of the model. Calling these files in a TensorFlow session will make run the model. For the code structure, we will proceed as following:

1. Class Network
    - contains a class Network which load the neural network with the check point files
    - resizes the image in correct dimensions to make correctly run the model
    - get in output the predicted boxes and confidences
2. Rectangles treatment
    - contains a method `convert_to_rectangles` which converts the predicted in boxes in rectangles.
    - regroups rectangles of a same waste according to a certain overlap
    - contains a method which returns the list of coordinates of the detected rectangles and the image with the rectangles drawn for each type of waste
    - contains a method `rectangles_overlap(r1, r2)` which returns `True` if the two rectangles overlap at least with the input overlap, false else

3. Detection part

   - loops on all images and computes predictions and draws rectangles of detected waste for each image
   - stores the predictions in a dataframe as following:

File Name	Glass	Bags	Wood

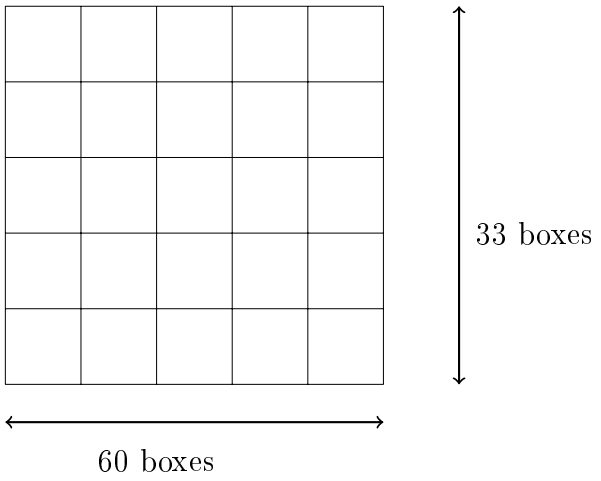
   - stores all the images with drawn rectangles in a new folder

4. Statistics

   - load a similar dataframe to the previous one containing the coordinates of the true/real waste bounding boxes
   - compare both dataframe and computes classical statistical tools: precision, recall, F1-score.

3.2.2 Bounding boxes

Now let's explain the part concerning the bounding boxes. The model divided the image in 1980 cells: 60 boxes in width and 33 boxes in height:



When a TensorFlow session will be run on an image, the model will loop on the 1980 cells. For each cell it will predict a rectangle for **each type of waste** with a certain confidence. The confidence is a value comprised strictly between and 0 and 1 which tells us about the confidence of the prediction. For example, if the model detects a glass with a confidence of 0.0001, we can consider that the prediction is false, whereas for a confidence of 0.8 we can consider it correct.

Running a Tensorflow session is made by calling the method `run` of a TensorFlow session. In this case of the given model, this method gives two outputs that we are going to describe: `pred_boxes` and `pred_confidences`. These two outputs are actually **tensors**.

`pred_boxes` is an array that contains the bounding boxes coordinates for each cell.

$$pred\_boxes = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ \vdots & \vdots & \ddots & \vdots \\ a_{1980,1} & a_{1980,2} & a_{1980,3} & a_{1980,4} \end{pmatrix}$$

The four columns correspond to the four coordinates of each cell. For example, the term `A(1,1)` correspond to the first coordinate of the bounding box of the first cell.
However the four coordinates of the bounding boxes must be converted into rectangles. Indeed, in Python rectangles are considered as 4 coordinates: the two first correspond to the left bottom angle and the two last to the right top angle. Here, the four coordinates of the bounding boxes don't contain directly such information. Lets introduce the link between bounding boxes coordinates, and predicted rectangles coordinates.
Let's denote the four coordinates of the bounding boxes by `bbox[0]`, `bbox[1]`, `bbox[2]`, `bbox[3]`. We have:

$$\begin{cases} c_x = \lfloor bbox[0] \rfloor + \frac{r}{2} + r \cdot x \\ c_y = \lfloor bbox[1] \rfloor + \frac{r}{2} + r \cdot y \\ w = bbox[2] \\ h = bbox[3] \end{cases}$$

Where:

- r is a parameter called region size
- x is the abscissa of the grid in the image
- y is the ordinate of the grid in the image

`pred_confidences` is an array that contains the confidence of the prediction of a type of waste for each cell.

$$pred\_confidences = \begin{pmatrix} b_{1,1} & b_{1,2} & \dots & b_{1,63} \\ b_{2,1} & b_{2,2} & \dots & b_{2,63} \\ \vdots & \vdots & \ddots & \vdots \\ b_{1980,1} & b_{1980,2} & \dots & b_{1980,63} \end{pmatrix}$$

The 1980 lines correspond to the cells and the 63 columns correspond to each type of waste.
For example, if `b[2,7] = 0.6`, it means that in the second cell, the algorithm predicts a waste of type 7 with the confidence of 0.6.

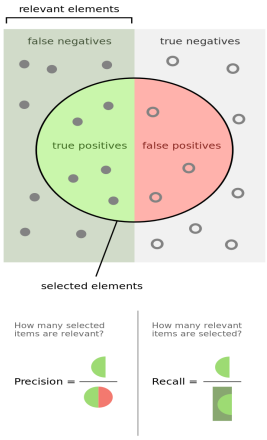
Summing all of this, for each cell there is a predicted rectangle with a confidence value for each type of waste. Here, we introduce a parameter called `min_confidence`. This parameter is a threshold: we will consider a prediction on a cell as correct if the associated confidence is higher than the minimum confidence. If it is not the case, the prediction will be considered as false. This parameter will be optimized later in our work.

3.2.3 Results storing

Now that we understood how a prediction works on a picture, let's explain how results are stored. The prediction is looped on all the pictures. Then, a dataframe (like described before) is built in order to store the predictions for each file (image) and for each type of waste.

The real rectangles are stored in a similar dataframe. By ordering the dataframe by file name (in alphabetical order), each line of both dataframe can be compared. This comparison directly leads to statistical analysis.

3.2.4 Statistical analysis



As we are in the case of a **multi classification problem**, we will compute classical statistical tools: precision, recall and F1-score. Firstly, let's make a little reminder about these three quantities.

The precision answers to the question: How many selected items are relevant?. In other terms, the precision is the percentage of **correct predictions**. The recall answers to the question: How many relevant items are selected?. In other terms, the recall is the percentage of correct predictions among relevant items [6]. Let's now mathematically introduce precision, recall and F1-score.

The true positive are the predictions that are correct. For example, if the model predicts a bag and that the object is actually a bag, the measure it's a true positive.

$$precision = \frac{TP}{|predictions|} \tag{1}$$

$$recall = \frac{TP}{|items|} \tag{2}$$

The F1-score is just an harmonic mean of precision and recall:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{3}$$

Returning to our algorithm, we can know explain how we will compute these statistical values.
These three values will be computed for each class. For the global precision for example, we will calculate an **average weight** of the precision of the three classes.
Regarding our dataframe, it is very easy to count the total number of predictions or the total number of relevant items. Indeed, it just consists in counting all the elements of each column in both dataframes. For example, if we want to count the total number of predictions for the bags, we just have to count the total number of rectangles list in

the column "bags" of the predictions dataframe. It remains to count the number of true positive for each class. Before entering in the details of the algorithm that will count the number of true positive, let's just explain how we consider the overlap between two rectangles

In order to analyze the overlap between two rectangles, we have to take into account the Intersection of Union, also called IoU. It represents the area of the intersection between two rectangles.

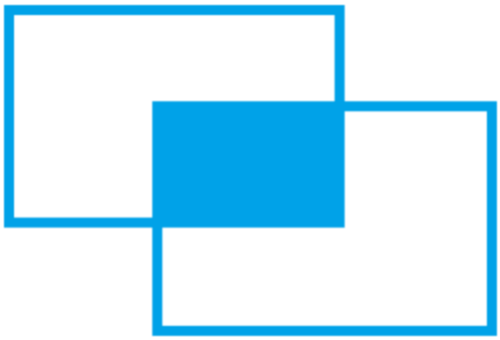


Figure 1: Intersection of Union between two rectangles

This tool is essential in our algorithm. Indeed, it is the most relevant one in order to evaluate if a prediction can be considered as true positive or not. For example, if IoU between a predicted box and a true box is equal to 0.90 (90% common area), we can consider the prediction as a true positive. However, for an IoU smaller than 0.5 for example, it would be maybe better to consider it as false. It means effectively that our model doesn't localize very well objects. As a remark, we can see that in our study case, the IoU considered can be different between the different type of waste. Indeed for big waste (like bags), the IoU between prediction and reality begins to be satisfactory from 0.5. For smaller waste, like glass, an IoU of 0.2 can be efficient: indeed for smaller waste, small IoU don't show that the prediction is negative.

Let's now describe the algorithm that will count the number of True Positive. As we said before, to do that, we will loop on each line of both dataframe. We are sure that a same index line points to the same file in both dataframe because we reordered them by file name in alphabetical order. Now let's describe the algorithm that will count the true positive for each class. We will describe the algorithm that compares two lists of rectangles. Indeed, we loop on files (actually on each line of both dataframes). Then for each line of both dataframe, we compare each similar column. As we want to count only true positives, we compare predicted bags rectangles with true bags rectangles, predicted glass rectangles with true glass rectangles and predicted wood rectangles with true wood rectangles. Let's describe the part of the algorithm which compares these two types of lists. We denote $L_1 = [[x_1, y_1, x_2, y_2], ..., [x_1, y_1, x_2, y_2]]$ and $L_2 = [[x_1, y_1, x_2, y_2], ..., [x_1, y_1, x_2, y_2]]$ two lists of rectangles. We will consider that L_1 contains the predicted rectangles and L_2

contains the real rectangles.

Algorithm 1: Count True Positive

```
Result: Number of true positive in the list L_1
if L_1 and L_2 are not empty then
    for rect_1 in L_1 do
        pred ← 0;
        for rect_2 in L_2 do
            if rect_1 and rect_2 overlap then
                TP ← TP + 1; pred ← 0;
                if pred > 1 then
                    pred ← pred + 1;
                end
            end
        end
    end
end
return TP
```

The value `pred` is useful to count the number of predictions done by $rect_1$. Indeed, at the beginning of the function, the number of predictions are counted as the length of the list of predicted rectangles. However, a predicted rectangle can correspond to many true waste. That is why it is important to count it as multiple True Positive, but also to add it as multiple predictions. Now having the number of true positive, precision, recall and F1-score can easily be computed.

3.3 Dataset

3.3.1 VoTT

The dataset (testset) on which we will test the neural network is composed of **334 pictures**. Each picture can contain none or all of the types of waste we want to detect. For each picture, we will manually tag waste. We will use **VoTT** (Visual Object Tagging Tool. This tool is an open source app used for labeling images in computer vision projects. For each picture we draw a rectangle around the waste. Then, the bounding boxes of these rectangles are saved in a JSON file. All of these JSON files will be very useful in order to build a dataframe containing the coordinates of the real rectangles waste. This dataframe will have a similar structure to the dataframe containing the predictions. We can show an example of image tagging.

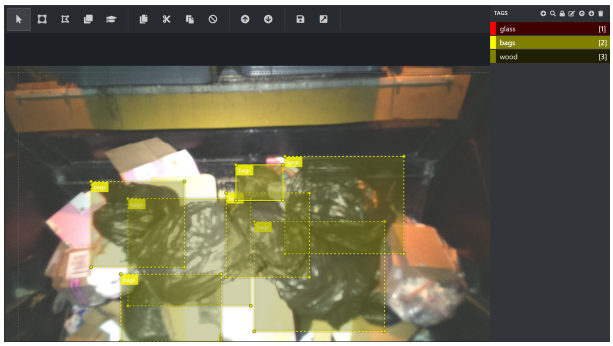


Figure 2: Example of use of VoTT for tagging bags waste

3.3.2 Labelling the data

As we mentioned, for each image, after drawing all rectangles, a JSON file is saved. This file contains several important information specific to each. An example is shown in figure 2.


```
{
    "asset": {
        "format": "jpeg",
        "id": "0cce7d06ecd4384b2ab03cf7a92b7a26",
        "name": "xxx_xxx_EG946PQ_xxx(303).jpeg",
        "path": "file:C:/Users/Thomas/Desktop/Cours%20ing%C3%A9nieur/Projet%20de%20semestre/photos/testset/xxx_xxx_EG946PQ_xxx(303).jpeg",
        "size": {
            "width": 1920,
            "height": 1280
        },
        "state": 2,
        "type": 1
    },
    "regions": [
        {
            "id": "GvOt4qdam",
            "type": "RECTANGLE",
            "tags": [
                "glass"
            ],
            "boundingBox": {
                "height": 115.51673944687045,
                "width": 70.02129304073715,
                "left": 754.5916813166828,
                "top": 896.687107669214
            },
            "points": [
                {
                    "x": 754.5916813166828,
                    "y": 896.687107669214
                },
                {
                    "x": 824.61297435742,
                    "y": 896.687107669214
                },
                {
                    "x": 824.61297435742,
                    "y": 1012.2038471160844
                },
                {
                    "x": 754.5916813166828,
                    "y": 1012.2038471160844
                }
            ]
        },
```


Figure 3: Structure example of the JSON file generated by VoTT

Here the most important information are:

- the file name (name)
- the image size. When the neural network, each image is re sized. It means that the predicted bouning boxes correspond to the re sized image. So, we have to normalize our real bounding boxes, and then re size them according to the "re size factor" of the neural net.
- regions, which contain all of the bounding box. For each region, there is a tag. A tag can takes only three values (glass, bags, wood). A region contains also a bounding box. This bounding box is composed of **4 key coordinates**: the height (height of rectangle), the width (width of rectangles), left (abscissa of bottom left corner of rectangle) and top (ordinate of bottom left corner of rectangle)

Now, labeling the data means to create a DataFrame containing the real bounding boxes for each file. To do that, we will code a Python algorithm which will loop on each JSON file. After computing this algorithm, we just have to add in the dataframe a line containing

Algorithm 2: Get the bounding boxes for one image file

```
Result: List of bounding boxes for each type of waste
for each JSON file do
    glass ← [];
    bags ← [];
    wood ← [];
    for region in file.regions do
        if region.type is "glass" then
            ad coordinates to glass;
        end
        if region.type is "bags" then
            ad coordinates to bags;
        end
        if region.type is "wood" then
            ad coordinates to wood;
        end
    end
end
return glass, bags, wood
```

the file name and the coordinates of the bounding boxes for each label.

3.3.3 Labels distribution

Regarding the labels distribution, the testset is clearly **unbalanced**.

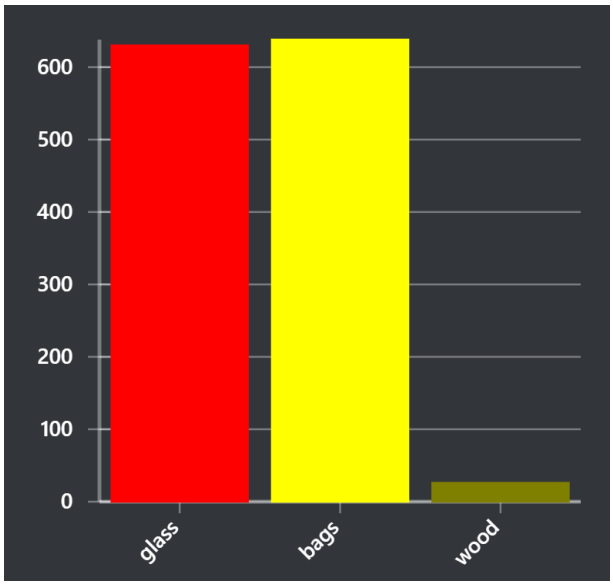


Figure 4: Labels distribution in the testset

Moreover the training set was also unbalanced like that with less wood than the two others waste types. If the results show lower performances for wood, it will be certainly caused by that.

3.4 Results

After running the different parts of the code, we can obtain the results. Firstly, we had to fix some initial values for parameters. Some of them will be optimized in the next section. The different parameters are:

- `min_confidence`, the minimal confidence for which we consider a prediction as reliable or not (the threshold confidence)
- `overlap threshold`, this overlap factor corresponds to the overlap between the rectangles of all cells during the image decomposition in grid. Indeed, we remind that an image is divided into 1980 cells/grids. However, a same waste can correspond to rectangles of multiples cells. That is why we fix an overlap factor for which we regroup rectangles in a same big rectangle.
- `overlap`, this overlap correspond to the overlap between a predicted rectangles and a true rectangle. It is very important, because if a predicted rectangle overlap with a real rectangle at least with this overlap factor, we consider that the two rectangles represent the same waste. Therefore, the prediction will be considered as a true positive.

Firstly we decided to fix these values:

- `min_confidence = 0.2`
- `overlap threshold = 0.5`
- `overlap = 0.25`

The following results have been obtained:

		precision	recall	F1-score
0	Glass	0.587156	0.203175	0.301887
1	Bags	0.871053	0.518809	0.650295
2	Wood	0.255319	0.461538	0.328767
3	Weighted AVG	0.720462	0.363988	0.474208

The results are quite satisfactory. As predicted, we get poorer results for wood. This is due to an **unbalanced dataset**. Moreover, by changing the parameters values, the precision can be improved. Let's compute a parameters optimization in order to increase as well as possible the precision. We will also try to keep a correct recall.

3.5 Parameters optimization

In this section, we will optimize two parameters: `min_confidence` and `overlap`. We will do a grid search for optimal parameters. This method is very similar to a cross validation. However, as the model is pre-trained, and not trainable because we have only access to the checkpoints files, the subsets of the cross validation can't be trained. This makes the major difference with our method.

3.5.1 Minimal confidence

Firstly we will optimize the minimal confidence, i.e. the confidence treshold for which we consider a prediction as positive or not. Our grid search is built as following: `[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]`. We search for the value that optimizes the precision.
We will plot two graphs: recall and precision function of confidence, and **RPC Diagram**, which is recall function of precision for different values of confidence.

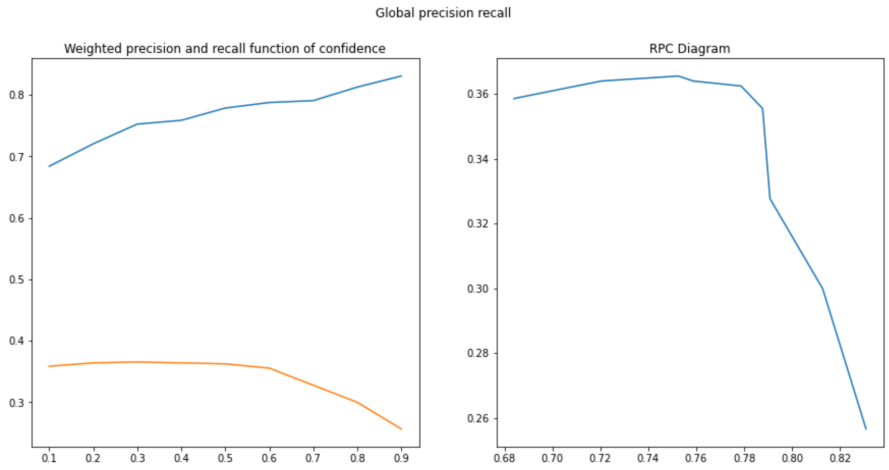


Figure 5: On the left graph, the precision is in blue, the recall in orange

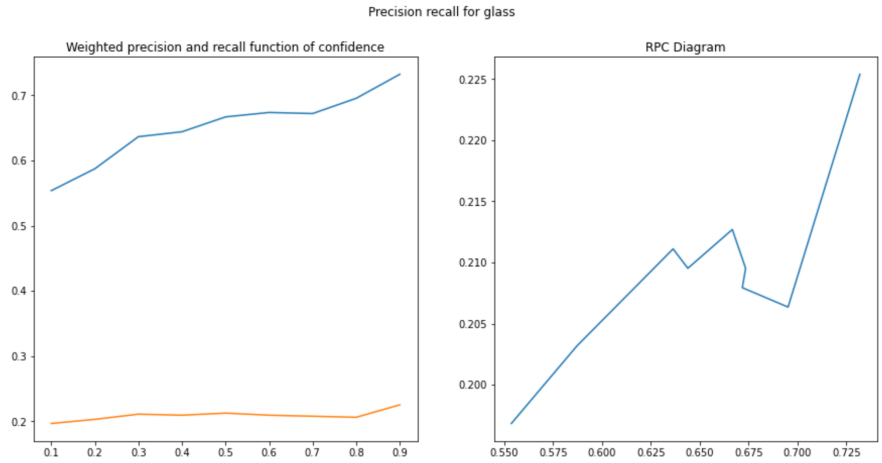


Figure 6: Precision recall for the glass

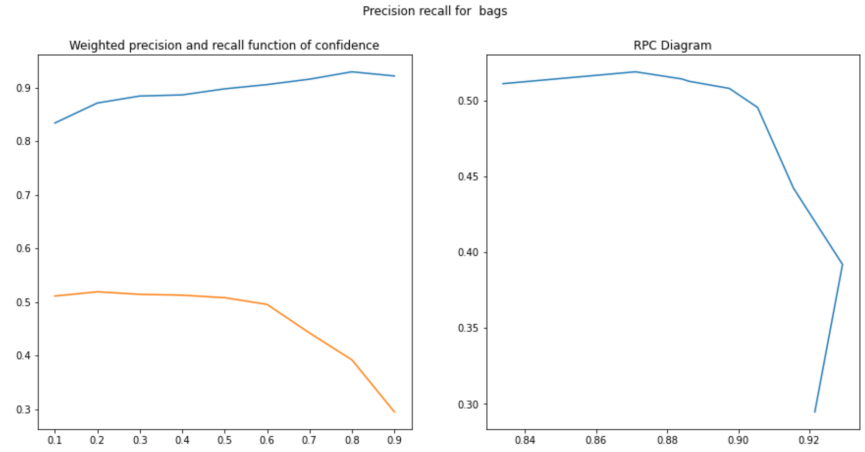


Figure 7: Precision recall for the bags

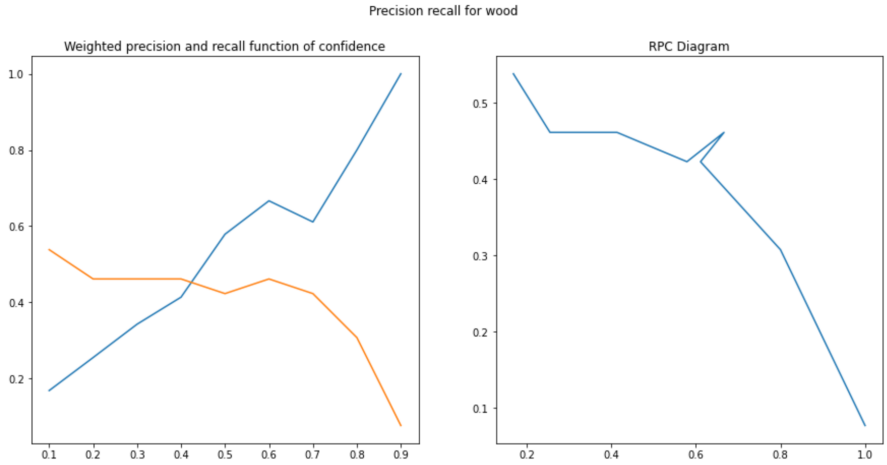


Figure 8: Precision recall for the wood

We pick 0.5 for the confidence. Indeed, increasing the confidence makes also increasing the precision. However, we would like to obtain a correct recall. For this reason, we chose 0.5, which is a classical value for this parameter in computer vision. Now let's optimize the overlap value.

3.5.2 Overlap

Here, we build again a grid search on overlaps. The grid is the same than the previous one. Let's plot recall and precision functions over this overlap grid:

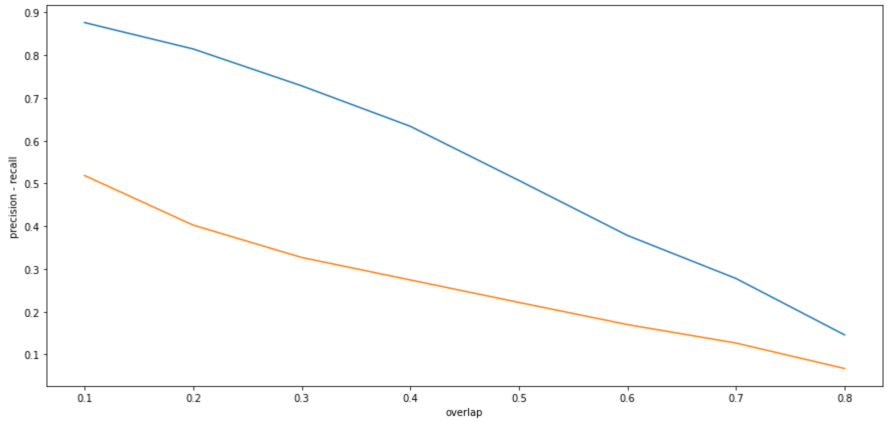


Figure 9: Again, the precision is in blue, the recall in orange

Increasing the overlap provokes a decrease for precision and recall, i.e. in performance. That is why, we will pick 0.2 for the overlap.

The optimal values are: `0.5` for the minimal confidence and `0.2` for the overlap.

3.6 Final results

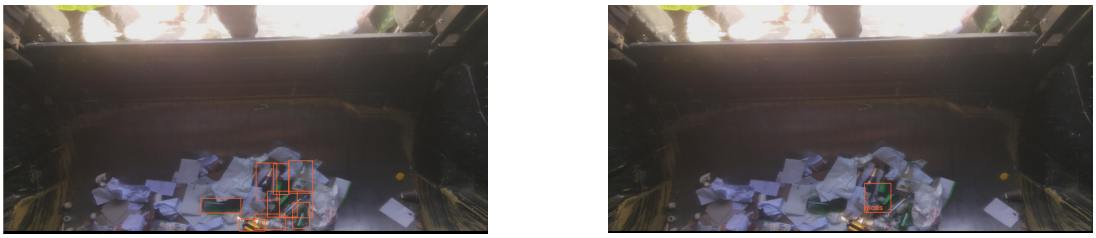
After optimization, the results are recaped in what follows: The results after optimization

		precision	recall	F1-score
0	Glass	0.726457	0.257143	0.379836
1	Bags	0.910995	0.545455	0.682353
2	Wood	0.578947	0.423077	0.488889
3	Weighted AVG	0.814479	0.402628	0.531182

are quite satisfactory. The global precision is good (81%) and the F1-score is about 53%. However, the recall is very low for glass, and the precision for wood is not that much high. This is due to an unbalanced train dataset (less wood than the two other types). We will try to overpass this problem by applying data augmentation in the next part.

3.7 Results on images

We can now show results on real images, by comparing human tagged images with predicted tagged images:



(a) (b)

Figure 10: Human tagged on left, predicted boxes on right for glass

(a) (b)

Figure 11: Human tagged on left, predicted boxes on right for glass

(a) (b)

Figure 12: Human tagged on left, predicted boxes on right for wood

On figures (a), there are the bounding boxes made by humans (real waste). On figures (b), we can find the predicted bounding boxes by the algorithm.
Thus we can see that the detection is quite satisfactory. However we can notice a low recall for glass.

4 Yolo algorithm

In this part, we are going to implement our own model. The model we chose is **YOLO (You Only Look Once)**. There are different models but YOLO presents some interesting advantages: it is very fast (45 frames per second) and it understands generalized object representations (the neural network can be trained on real world images). The last point is essential. Our dataset will mainly be composed of specific images, captioned by a camera in a truck. These images are taken from the real world. The fact that YOLO works well on this type of images motivates us to pick this model.

4.1 Datasets

Here, we will use a classical approach in order to build our model. We will use two datasets. One of them will obviously be the one with the waste we have to classify. The other, will be a lighter one, that will be used only to check if our code is consistent. Indeed, it is important to have a reference in order to be sure that our model will work on data. In other words, this step is only used to be sure that, in the case of bad results, this is not due to a mistake in our model.

4.1.1 Dataset to check the code validity

The dataset to check the validity will be a classical one, not taken from the real world. I used Kaggle datasets to extract one. I took the **Malaria** dataset [3]. This dataset contains pictures in which we can find different type of cells: red blood cells (RBCs), leukocytes, trophozoites, rings, etc. The dataset is clearly unbalanced: more red blood cells than trophozoites for instance (this is because RBCs are uninfected cells whereas trophozoites are infected ones). To test if our YOLO works well, we will try to make detection of RBCs and trophozoites. Here, our choice has been made arbitrary .

4.1.2 Trainset for the project

The dataset to train our YOLO model will be very similar to the previous one we used. However, we will keep the previous dataset to test our YOLO model, in order to have a common comparison basis. Our trainset will be again composed of images captioned by trucks. In this images, we can find bags, glass, wood or nothing among these 3 types of waste. We regrouped about 2959 images. These images have been labeled using the same software as previously, i.e. **VoTT**. After drawing a bounding box each time I find one of the three type of waste, I obtained JSON files that contain bounding boxes for the labeled images. We can note that this step is also efficient in order to remove useless data: indeed JSON files are generated only for images that effectively contain an object. We get a total of 1889 JSON files, which eliminates about 1000 useless data.

Now, we can make some data visualization. This step is useful in order to observe how our dataset is formed and distributed. I use this step especially to check if the dataset in **unbalanced**. It will allows us to predict bad results on a particular label for example. The data visualization clearly shows us that the dataset is unbalanced: indeed, there are

a lot of bags, a satisfactory number of glass, but a very small collection of wood. We can predict bad results on wood, and maybe good results for bags.

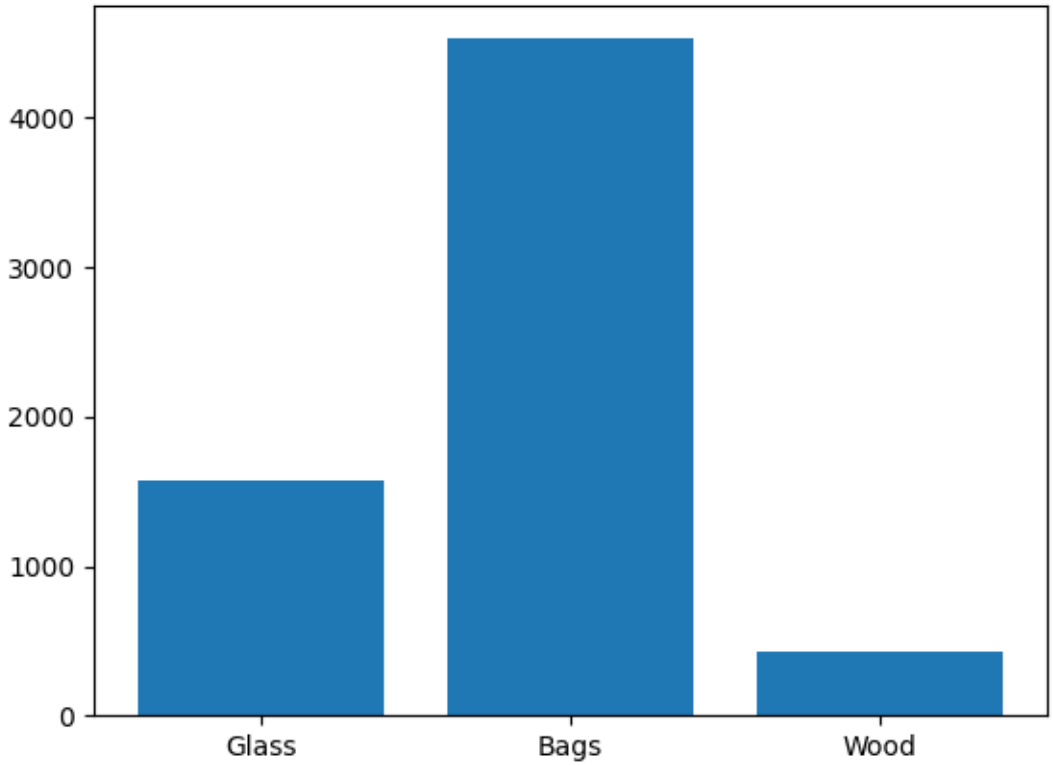


Figure 13: Data distribution for the trainset

4.2 The model

Now that our two datasets are prepared, we can start implementing the model. In this section, we are going to present the basics of YOLO model, directly taken from the original paper [4].

4.2.1 Introduction

The most recent approaches like R-CNN for example, use region proposal methods to generate bounding boxes. Then a classifier is run on the potential candidates. After this classification step, post-processing refines the bounding boxes and eliminates duplicates if it is needed. The major problem of such a pipeline is that it is very slow and hard to optimize. YOLO works differently: based on the **you only look once system**, a single convolutional neural network is run on the full image and directly makes the optimization. As we can see, the pipeline is much lighter, which presents several advantages. First, the model runs very fast: 45 frames per second on good GPUs (Titan X for instance).

Secondly, YOLO uses the global understanding of the image: it is not focused on specific regions of the image. The model tries to extract contextual information about classes. Finally, a main advantage of YOLO is that it understands generalized representations of objects. Thus, the model can be executed on real world objects, which is particularly interesting for our project.

4.2.2 Detection principle

As it is a Deep Learning approach, we will use **tensors**. The input is the image. The YOLO system divides it into a grid. For squared pictures, we can take $S \times S$ grid. For rectangular pictures, we can take $S \times T$ grid. As our pictures have a dimension of 1920×1080, we will use rectangular grid approach. Then, each grid is scanned by the model. The cell responsible to detect an object is the one that contains the center of the object.

Now let's see how YOLO treats a bounding box. As a recall, a bounding box is the rectangle that frames the detected object. A bounding box has 5 predictions: x, y, w, h and the confidence. The two first parameters x, y are the coordinates (abscissa and ordinate of the center of the rectangle). It is important to note that this coordinate is relative to the bounding box in which it is located. Some operations must be done in order to scale these coordinates in the whole picture. Then w and h are the width and the height of the bounding box. As a remark, let's discuss the chosen representation of a bounding box with the four first parameters. There are other representations: for example four coordinates, two for the left bottom corner and the two others for the top right corner. One benefit of the YOLO representation is that if we want to translate a rectangle, we only need to change the value of the center coordinates (the width and height are unchanged). Finally, the confidence is an indicator of the quality of the prediction. It is a float comprised between 0 and 1. The more the confidence is close to 1, the more the detection is likely to be correct.

We discussed about tensors. We explained that the output is the image, which will be divided in a grid by the system. Then, the system will scale among all the grid cells and computes detection. The output tensor will therefore be more complex. Let's take our study case, with a rectangular approach ($S \times T$ cells). The predictions will be encoded in a tensor that presents the following dimension:

$$S \times T \times (B \times 5 + C)$$

S and T are the numbers of grid cells. B is the number of objects we want to detect per cells. In our case, waste have a certain volume. For this reason, it will be very rare that there are more than 2 waste per cell. We can fix $B = 2$, but this is chosen arbitrary. Besides, 5 stands for the 5 predictions of a given bounding box. Finally C is the number of classes we want to detect (here 3). So, in our case, if we take 16 cells in width and 12 cells in height for example, the tensor will have a dimension of: $16 \times 12 \times (10 + 3)$.

4.2.3 Network design

Now that we understood the principle, we still need a neural network model in order to make the predictions and to connect layers. There a several models that can be chosen. Here, we will take the model presented in the paper [4]. The network is largely inspired by the GoogLeNet model. It has 24 convolutional layers followed by 2 fully connected layers.

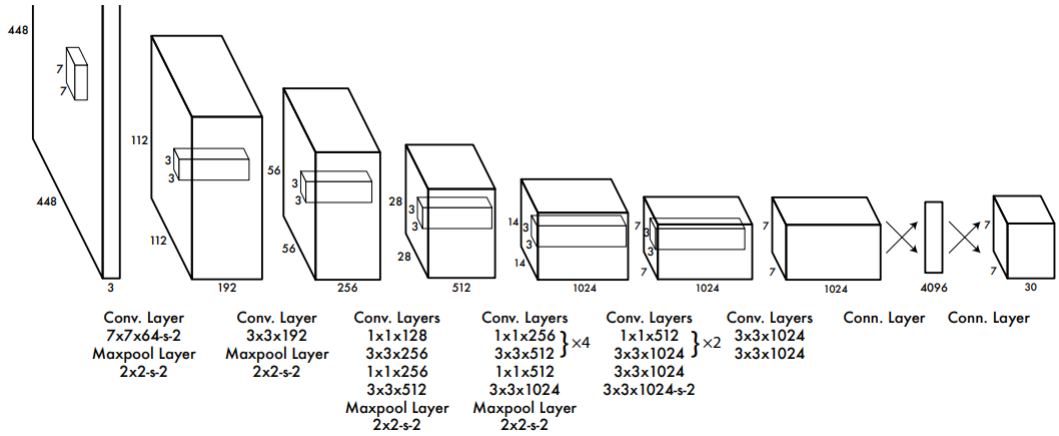


Figure 14: Architecture presented in the YOLO paper

However certain things can be changed in this model. Firstly, in the YOLO v2 paper, it is advised to not fully connect the two last layers. For this reason, we will not do it in our model. Moreover, we will use **ResNet**. The main advantage is that they present better results in general.

4.2.4 Training and inference

Now that we have our model, we can think about the training part. In the training part, the input will be the image with the corresponding bounding boxes. During the training, the objective is to optimize the following loss function:

$$
\begin{aligned}
&\lambda_{coord} \sum_{i=0}^{T\times S} \sum_{i=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2\right] \\
&+ \lambda_{coord} \sum_{i=0}^{T\times S} \sum_{i=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(w_i - \hat{w}_i)^2 + (h_i - \hat{h}_i)^2\right] \\
&+ \sum_{i=0}^{T\times S} \sum_{i=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
&+ \lambda_{noobj} \sum_{i=0}^{T\times S} \sum_{i=0}^{B} \mathbb{1}_{ij}^{\text{obj}} (C_i - \hat{C}_i)^2 \\
&+ \sum_{i=0}^{T\times S} \mathbb{1}_{i}^{\text{obj}} \sum_{c\in\text{classes}} (p_i(c) - \hat{p}_i(c))^2
\end{aligned}
$$

This loss function seems huge but it is only based on sum-squared error. Sum-squared error is used because it is easy to optimize [4]. Here, there are five sum-squared error functions: one for the center coordinates, one for the width and the height, two for the confidence (when an object is present or not) and one for the class vectors. The indicator function is used to take into account the term only if there is an object in the cell. Indeed, we will consider the sum-squared error over coordinates only if there is an object. However, the sum-squared error must be taken into account even if there is no object. Indeed, the confidence is the first parameter at which we will take attention, so it is important to check the confidence also if there are no objects. As the confidence is less important when there are no objects (compared to if there is an object), we put a parameter λ_{noobj} in front of this element. This parameter will take the value 0.5 which will reduce the importance of this term.

For the training part, the activation function that we will use is the **leaky rectified linear activation (leaky ReLU)**, as suggested in the paper [4]:

$$f(x) = \begin{cases} 0.1x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases}$$

The inference (testing) is very simple. After training the model, we load it with the weights we obtained. We will do it by restoring the **checkpoints files** generated. Then we apply this neural network to our testset. It will give the tensors with same dimensions in output. This is how we will get our predictions.

4.2.5 Recap of parameters

Like in the first part of the project, here again we face many parameters. Let's do a summary of this parameters:

- grid parameters: how we divide our image in entry. The dataset is only composed of rectangular images (1920×1080 pixels). This is the reason why we will change a little bit the grid compared to the YOLO paper. We will take 16 cells in width and 12 cells in height.
- confidence: like in the first part, the confidence parameter is a threshold that will fix the minimal confidence for which we will take into account a prediction. We will take a classical value (inspired from papers and by experience): 0.5.
- IoU (intersection of union): here again we have two overlap thresholds. We will have overlap between rectangles that seem to designate a same object. To fix this issue, we will use **Non Max Suppression** which is explained later. The second overlap threshold is present again in the metrics, between a predicted rectangle and a true one. Here we will take standard value, 0.5.

4.3 Data augmentation

As our dataset is clearly unbalanced and because of a real lack in wood objects, we decided to implement data augmentation. Again, this is a classical approach to solve this kind of

problems. Data augmentation are techniques that are used to increase the total number of data by modifying the original images (rotations, noise, lightness, symmetry, re-sizing, and so on). In this project we used different kind of such operations.

The libraries I used are numpy and openCV. Firstly, I implemented two Python functions: one to add noise thanks to clip method of numpy, and one to randomly change the brightness of the picture using again clip method. Then, I randomly made other operations on pictures. To perform that, openCV library is a rich tool. It provides methods that can compute symmetry on pictures. The method I used is called flip. I takes a code as an argument. In function of the value of this code, different symmetries are performed: vertical flipping, horizontal flipping, etc.

It is important to understand that these images won't be created in the dataset. There are randomly generated during the training part. Another remark is that we don't need to label this augmented data. Indeed, bounding boxes of modified pictures can be easily obtained from the original labels. As we know which symmetry is applied to the picture, we can effectively apply geometric symmetry formulas to obtain the new bounding boxes. We can show example of data augmentation. Let's illustrate it on one picture:

(a) Original image (b) Brightness variation (c) Contrast

Figure 15: Example of operations for data augmentation

4.4 Anchors

As we explained, the YOLO model will detect bounding boxes with 4 parameters (center coordinates, width and height). These bounding box are rectangles: a rectangle can be a squared, horizontal and vertical. An horizontal rectangle means that the width is higher than the height. A vertical one is the opposite: height higher than width. Detecting the form of a bounding box, and replace it by one that has a particular form (squared, vertical or horizontal) can help to optimize the detection. The new bounding boxes are called **anchors**. There are very useful because they take into account the form of the object we want to detect. For instance, in our case, bags are likely to be squared. Wood are often horizontal (larger width than height). For the glass, we are facing an issue: indeed, these objects are not oriented in the same way in all pictures. Applying anchors to this objects is not an easy task. We can fix the horizontal one (glass are more often horizontal than vertical).

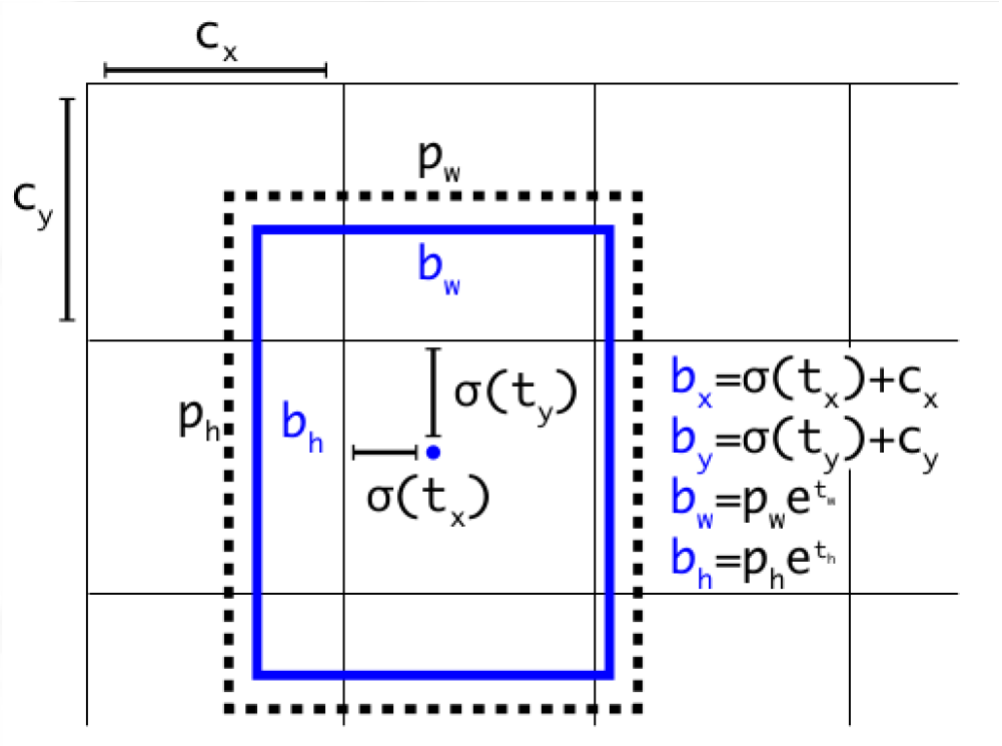


Figure 16: Anchors reformulation

The blue rectangle is the predicted one, and the black line stippled one corresponds to the anchor. In this example, the anchor model is a vertical bounding box.

4.5 Non max suppression (NMS)

It remains one important optimization: removing duplicated predictions. Indeed, predicting boxes and changing them into their corresponding anchors won't be enough: there will remain duplicated predictions. An object can be present in several cells. Thus multiple boxes can correspond to a same object. It will cause problem in metrics (precision can be higher than one for example), and the results will be difficult to be interpreted.

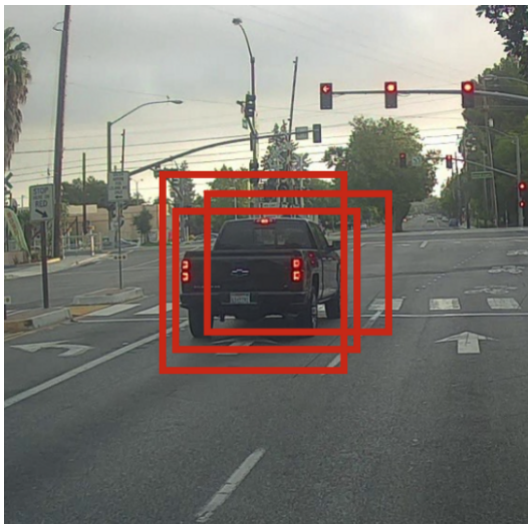


Figure 17: Multiple bounding boxes for same object problem

In the first part of the problem, we faced exactly the same issue. To solve this, we just implemented a code that grouped together rectangles that are likely to represent the same object. We will draw from this method and from a famous algorithm especially: **the Non Max Suppression (NMS)**. This algorithm aims to remove duplicates bounding boxes that represent the same object. It consists in keeping the bounding box with the higher confidence among all the bounding boxing designating a same object.

Algorithm 3: Non Maximum Suppression

Result: List of the selected bounding boxes with the corresponding confidences
Input : $\mathcal{B} = (b_1, \ldots, b_n)$ list of initial detection boxes, $\mathcal{S} = (s_1, \ldots, s_n)$ corresponding confidences, ϵ the IoU threshold

$\mathcal{D} \leftarrow ()$;
while $\mathcal{B}$ *is not empty* **do**
  $m \leftarrow$ argmax $\mathcal{S}$;
  $\mathcal{M} \leftarrow b_m$;
  $\mathcal{D} \leftarrow \cup\mathcal{M}$;
  $\mathcal{B} \leftarrow \mathcal{B} - \mathcal{M}$;
  for b_i *in* $\mathcal{B}$ **do**
    if $IoU(\mathcal{M}, b_i) \geq \epsilon$ **then**
      $\mathcal{B} \leftarrow \mathcal{B} - b_i$;
      $\mathcal{S} \leftarrow \mathcal{S} - s_i$;
    end
  end
end
return $\mathcal{D}, \mathcal{S}$

The output of the algorithm will be the selected bounding boxes with the corresponding confidences. The output is smaller or equal to the input: indeed we remove duplicates only if we find that there are ones. Regarding the example of the car, the output of the NMS algorithm will be like that:

Figure 18: Unique bounding box for the car after NMS

4.6 Code structure

The code of the model is published in the **Gitlab**. It follows the same logic as the YOLO model. Let's explain the different python files, their content and their role in the execution of the model.

- bounding box: a Python class used to easily create and modify bounding boxes. A bounding box a four key parameters: abscissa and ordinate of the center, width and height.
- clean trainset: I just implement this code to be sure that none image of the testset is present in the trainset. Indeed, testing on images that were used for the training part is useless
- data visualization: useful code to see how is the data distributed, especially to prevent an unbalanced dataset
- label img: I made this code in order to convert JSON files in text files. Indeed, we saw before that VoTT generates one JSON file per tagged image. These JSON files contain the useful bounding boxes. However, the major problem is that this kind of file, in this format, is not usable for YOLO model. This is why we need to make a conversion. This code converts each JSON file in a text file. The text file title will be the same as the image title, with the txt extension. In this text file, each line corresponds to a bounding box. The line starts by the class (bags = 0, glass = 1, wood =2) and then is followed by the four key elements (abscissa/ordinate of the center, width and height).
- config: config file containing model parameters (number of classes, labels classes, number of cells, batch size, ...)
- data augmentation: preparation of the labels and make the data augmentation

- model: contains the neural network that will be used (here the ResNet)
- train: training part of the code. Checkpoints file are saved after execution
- inference: testing part of the code
- metrics: contains useful metrics for rectangles treatment and also for evaluating the model (precision, recall and F1-score)

4.7 Results

Before presenting the results, it is important to say that the code has been tested on the Malaria dataset in order to check its validity. The YOLO model works quite good (F1 score of 0.76 for RBCs and 0.57 for trophozoites). We can now train and test our YOLO model on the waste images. Firstly we will show the results on the trainset, then on the testset.

4.7.1 On the trainset

On the trainset, the results are quite satisfactory, which is logical. The results are:

- bags: 0.93 precision, 0.85 recall
- glass: 0.68 precision, 0.48 recall
- wood: 0.91 precision, 0.84 recall

We can show the results on one picture of the trainset (predicted in dotted line):

Figure 19: Predicted bounding box on one image of the trainset

The predictions are very good, which is not surprising as far as we are testing on the trainset.

4.7.2 On the testset

On the testset, the results are not good at all. I won't mention results on wood because they are very low (as expected). The results are:

- bags: 0.35 precision, 0.17 recall
- glass: 0.1 precision, 0.05 recall

We can show the results on one picture of the testset (predicted in dotted line again):

Figure 20: Predicted bounding box on one image of the testset

The prediction is not very good (misses one bag) and one prediction for two objects.

4.7.3 Analysis

The results show a very low performance on the testset compared to the trainset. It directly leads us to the conclusion that our model is **overfitting**. There can be many reasons: bad choice in parameters, trainset not optimal, bad choice in data augmentation, bad normalization and optimization, etc. Regarding these low results, we can propose some techniques to largely improve the performances of the model.

4.8 Improvements

The first idea is that the dataset provided by Paprec is unusual: it is based on real world data. It doesn't affect YOLO: indeed YOLO model is capable of understanding very well real world data. However, the first improvement I can propose is to balance the dataset. It would be great to have more wood and glasses, especially wood. A too low number of wood features leads to bad results for this class.

Regarding the first part, we can see that the CortexIA model based on R-CNN presents good results. I think that YOLO will work better on this type of problem. However, the CortexIA model was pre-trained on 86.000 pictures, and then trained again with Paprec

data. That is why, I propose to work a little bit like in the YOLO paper [4]. Prepare a very large dataset (a little bit like the Pascal VOC dataset) that contains a large number of waste. Train our model on this dataset. Then make another training with our trainset (we can again use data augmentation). The results should be largely improved, and the approach will be closer to the one adopted in the paper.

Another remark is regarding the predicted pictures. Indeed, it seems that there is a lot of space in predicted boxes that is not related to the waste. A bag for example has a circular form. A bounding box framing this type of waste will contain space that is useless to learn the bag class. Same remark for the glass. Concerning the wood, the cages are squared, thus a bounding box is efficient and won't contain that much useless space. To fix this issue, we could for instance replace rectangles by ellipsis for bags and glass. It could maybe improve the performances of the model because the ellipsis will contain more useful information than the bounding box.

We can also evoke some methods that can improve performances: regularization and dropout [2].

References

[1] Arthur Le Denn. *Que fait Cortexia, la start-up suisse à qui Cédric Villani veut faire appel pour nettoyer Paris ?* L'usine digitale. https://www.usine-digitale.fr/article/que-fait-cortexia-la-start-up-suisse-a-qui-cedric-villani-veut-faire-appel-pour-nettoyer-paris.N939246. Mar. 2020.

[2] Ian Goodfellow et al. *Deep learning*. Vol. 1. 2. MIT press Cambridge, 2016.

[3] Kaggle. *Malaria dataset*. Malaria. https://www.kaggle.com/kmader/malaria-bounding-boxes. Consulted in January 2021.

[4] Joseph Redmon et al. "You Only Look Once, Unified, Real-Time Object Detection". In: *arXiv* (2016).

[5] Mohammad Saeed Rad et al. "A Computer Vision System to Localize and Classify Wastes on the Streets". In: *arXiv* (2017), arXiv–1710.

[6] Wikipedia. *Precision and recall*. Wikipedia. https://en.wikipedia.org/wiki/Precision_and_recall. Consulted in December 2020.